I/O workload characterization in MPI applications

I/O Bloopers

Yushu Yao

National Energy Research Scientific Computing Center

Lawrence Berkeley National Laboratory

Phil Carns

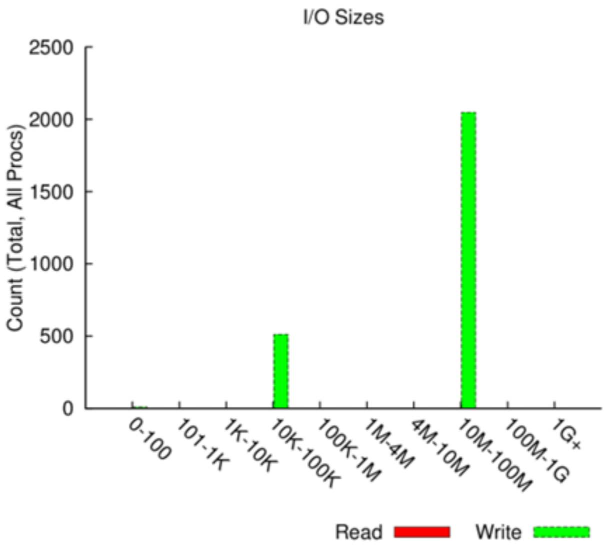
Mathematics and Computer Science Division

Argonne National Laboratory

How to find "I/O bloopers"

- Darshan can be used to identify a variety of I/O patterns that may lead to poor performance
- We'll show some examples from production applications at ANL and LBL to give some ideas of what to look for

Checking I/O expectations



- The Darshan job summary PDF includes:
  - a histogram of access sizes
  - a table of the most frequent exact access sizes
  - a table of file sizes
- This data is useful to confirm expected behavior for an application
  - In this case, there were 512 relatively small writes of 40 KiB each
  - That size corresponds to the file header size of the application (expected)
  - But there are only 129 files, why are there 512 headers?

Most Common Access Sizes

access size	count
67108864	2048
41120	512
8	4
4	3

File Count Summary

type	number of files	avg. size	max size
total opened	129	1017M	1.1G
read-only files	0	0	0
write-only files	129	1017M	1.1G
read/write files	0	0	0
created files	129	1017M	1.1G

Redundant Read Traffic

- **Scenario:** Applications that read more bytes of data from the file system than were present in the file
  - Even with caching effects, this type of job can cause disruptive I/O network traffic
  - Candidates for aggregation or collective I/O
- **Example:**
  - Scale: 6,138 processes
  - Run time: 6.5 hours
  - Avg. I/O time per process: 27 minutes

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	1299	1.1G	8.0G
read-only files	1187	1.1G	8.0G
write-only files	112	418M	2.6G
read/write files	0	0	0
created files	112	418M	2.6G

1.3 TiB of file data
500+ TiB read!

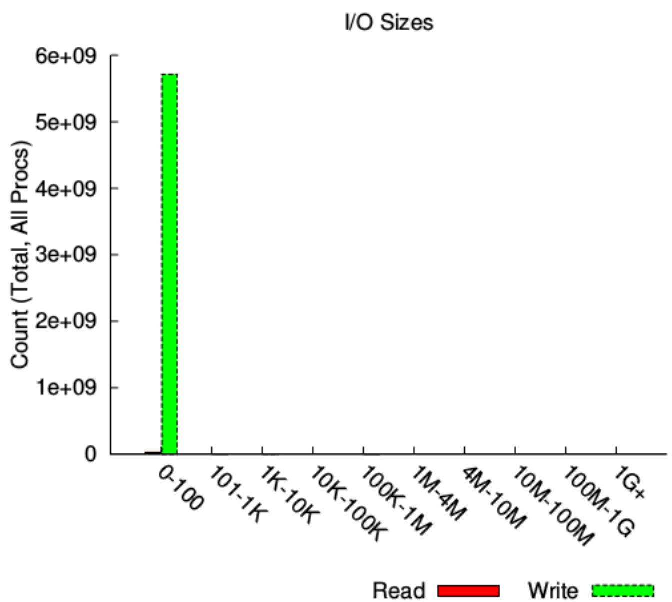
Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/	47161.47354	1.00000	575224145.24837	1.00000

Small Writes to Shared Files

- **Scenario:** Small writes can contribute to poor performance
  - Particularly when writing to shared files
  - Candidates for collective I/O or batching/buffering of write operations

- **Example:**
  - Issued 5.7 billion writes to shared files, each less than 100 bytes in size
  - Averaged just over 1 MiB/s per process during shared write phase

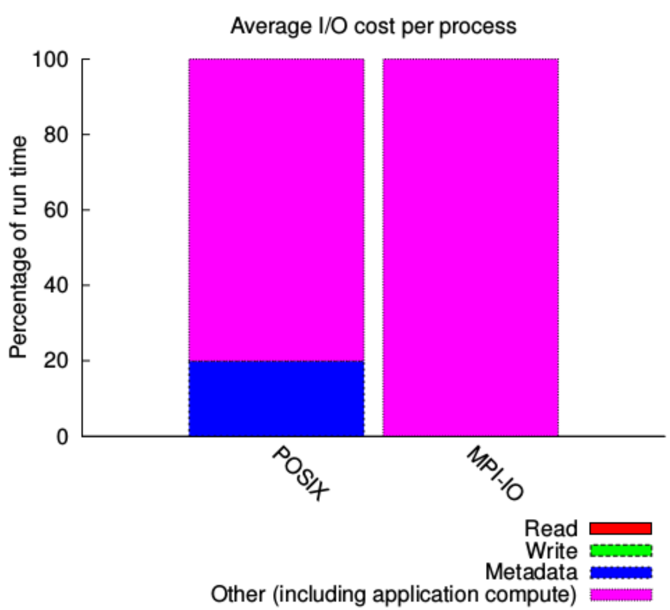


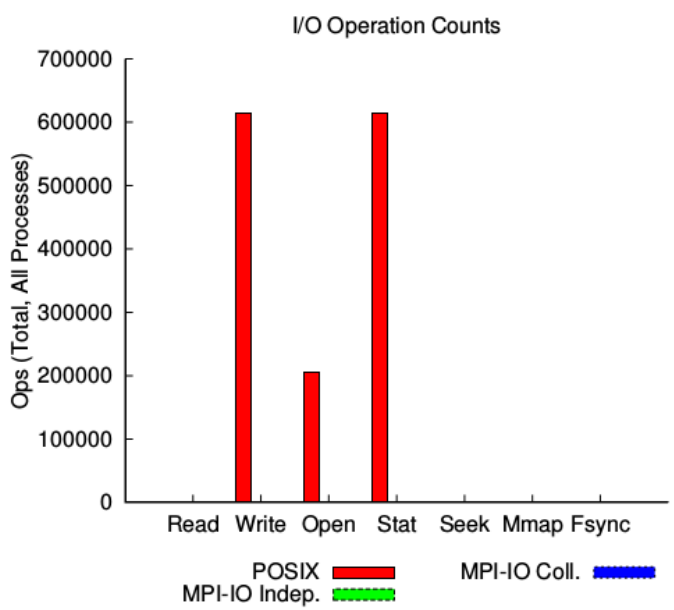
Most Common Access Sizes

access size	count
1	3418409696
15	2275400442
24	42289948
12	14725053

Time in Metadata Operations

- **Scenario:** Very high percentage of I/O time spent performing metadata operations such as open(), close(), stat(), and seek()
  - Close() cost can be misleading due to write-behind cache flushing
  - Candidates for coalescing files and eliminating extra metadata calls
- **Example:**
  - Scale: 40,960 processes for 229 seconds, 103 seconds of I/O
  - 99% of I/O time in metadata operations
  - Generated 200,000+ files with 600,000+ write() and 600,000+ stat() calls

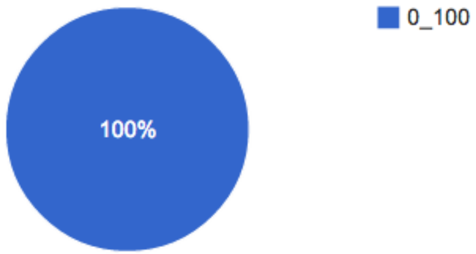




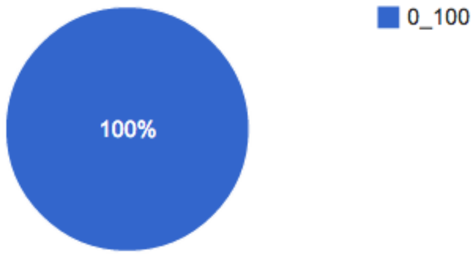
Using the wrong file system

Start	End	Wallclock (secs)	MB Read	MB Written	Estimated I/O Rate (MB/sec)	Estimated Percent Time Spent in I/O
07-18 22:36:19	07-19 05:32:16	24,957	217.0	640.2	0.11	31.47%

Number of Reads Per Size Range



Number of Writes Per Size Range

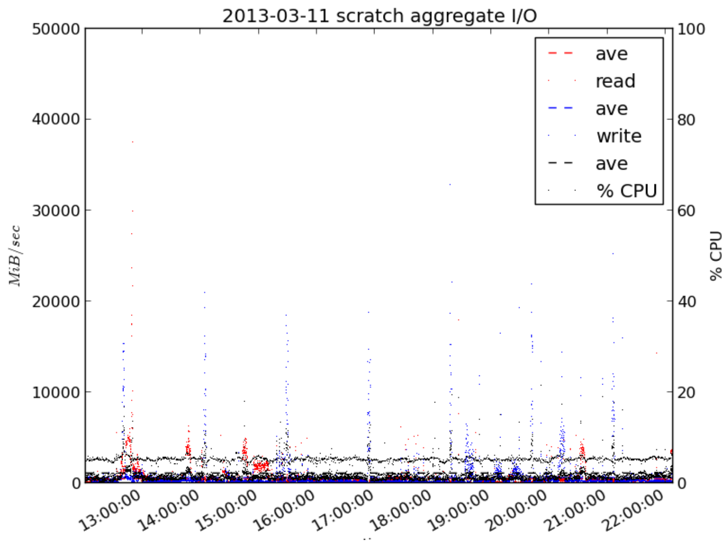


- Behavior:
  - A 40K core job uses MPI-IO to repeatedly write a small restart.dat file in /home filesystem
  - Many Open/Seek seek calls
- Problem
  - Spent 30% time writing only 600MB output
  - Using the wrong File System really hurts
  - Many metadata operations will hurt performance regardless of FS
- Suggestion
  - Use/scratch file system (higher bandwidth)
  - Reduce amount of metadata calls with collective buffering in MPI-IO

Yushu Yao

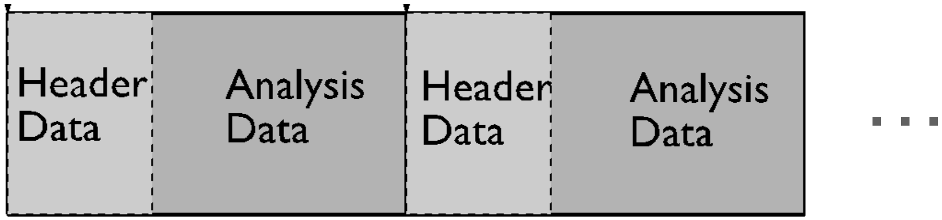
Checkpointing Too Frequently

- Behavior
  – A 300 node application writes a full checkpoint every hour, with good rate
- Problem
  – Spent 80% time in writing checkpoints
- Suggestion
  – Checkpoint less: Hopper has <1 node failure per day, so a 300 node job is expected to have a node failure only every 20 days. Checkpointing less frequently.



Yushu
Yao

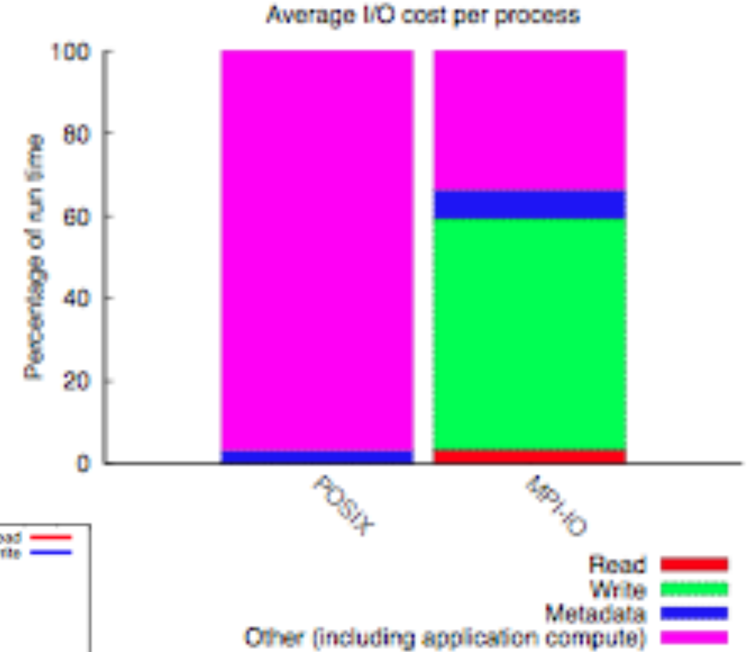
Performance Debugging: An Analysis I/O Example



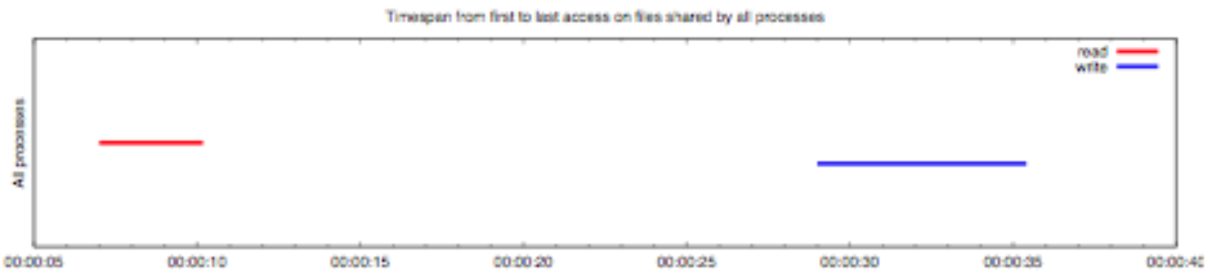
- Variable-size analysis data requires headers to contain size information
- Original idea: all processes collectively write headers, followed by all processes collectively write analysis data
- Use MPI-IO, collective I/O, all optimizations
- 4 GB output file (not very large)
- Why does the I/O take so long in this case?

Processes	I/O Time (s)	Total Time (s)
8,192	8	60
16,384	16	47
32,768	32	57

An Analysis I/O Example (continued)



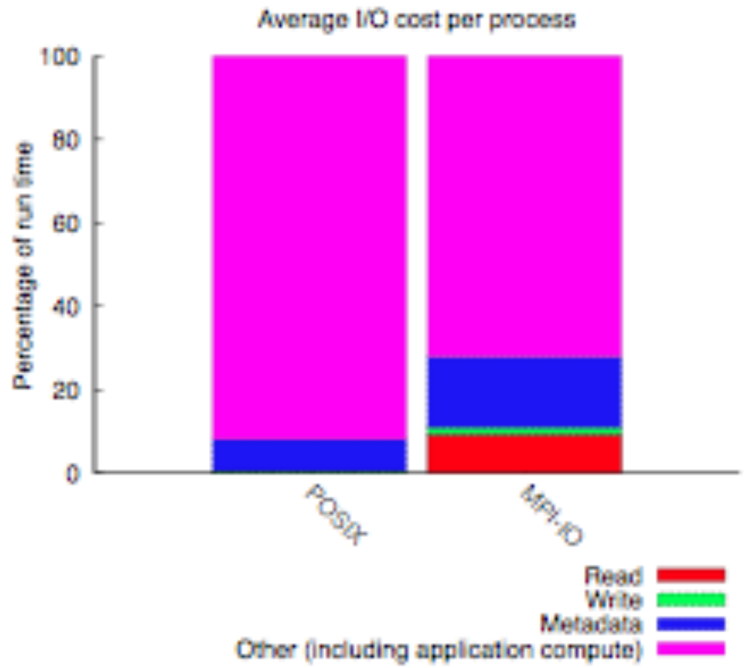
- **Problem**: More than 50% of time spent writing output at 32K processes. Cause: Unexpected RMW pattern, difficult to see at the application code level, was identified from Darshan summaries.
- What we expected to see, read data followed by write analysis:



- What we saw instead: RMW during the writing shown by overlapping red (read) and blue (write), and a very long write as well.

An Analysis I/O Example (continued)

- **Solution**: Reorder operations to combine writing block headers with block payloads, so that "holes" are not written into the file during the writing of block headers, to be filled when writing block payloads
- **Result**: Less than 25% of time spent writing output, output time 4X shorter, overall run time 1.7X shorter
- **Impact**: Enabled parallel Morse-Smale computation to scale to 32K processes on Rayleigh-Taylor instability data



Processes	I/O Time (s)	Total Time (s)
8,192	7	60
16,384	6	40
32,768	7	33

This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract Nos. DE-AC02-06CH11357 and DE-AC02-05CH11231 including through the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization.